УДК 519.7

В.В. Подымов, У.В. Попеско

ВЕРИФИКАЦИЯ ПРОГРАММНО-КОНФИГУРИРУЕМЫХ СЕТЕЙ ПРИ ПОМОЩИ СИСТЕМЫ UPPAAL

V.V. Podymov, U.V. Popesko

UPPAAL-BASED VERIFICATION OF SOFTWARE-DEFINED NETWORKS

В последние несколько лет активное развитие получили программно-конфигурируемые сети (ПКС) – особый вид компьютерных сетей, в которых все коммутирующие устройства имеют централизованное управление. В статье изучены задачи формального описания и верификации ПКС. Для описания ПКС использована библиотека элементов UML в редакторе диаграмм Dia. Для верификации ПКС использовано программно-инструментальное средство UPPAAL. Основной результат исследований – разработка транслятора, позволяющего по диаграмме сети получить ее модель для верификации в виде сети конечных временных автоматов. Корректность трансляции строго обоснована. Проведен ряд экспериментов, показывающих применимость предложенного метода верификации для проверки свойств поведения ПКС, специфицированных посредством формул темпоральной логики реального времени.

ПРОГРАММНО-КОНФИГУРИРУЕМАЯ СЕТЬ; ВЕРИФИКАЦИЯ; ВРЕМЕННЫЕ АВТОМАТЫ; ТЕМПОРАЛЬНАЯ ЛОГИКА; UPPAAL.

A lot of efforts were made in the last few years in the area of software-defined networks (SDN) – a special kind of computer networks in which the switching device control is fully centralized. This paper investigates the problems of formal description and verification of SDN as a real-time system. We provide a UML-based description of SDN, using the UML diagram editor Dia. To verify real-time properties of SDN, we use a well-known model-checking tool UPPAAL. The main result of the research is an approach for SDN verification, based on translation of SDN description into network of timed automata. Translation correctness is formalized and proved. A number of experiments were done to show that the approach can be used to verify real-time properties of SDN specified as TCTL formulae.

SOFTWARE-DEFINED NETWORKS; VERIFICATION; TIMED AUTOMATA; TEMPORAL LOGIC; UPPAAL.

Идея программно-конфигурируемых сетей (ПКС) сформулирована специалистами университетов Стэнфорда и Беркли в 2006 г. [1] В таких сетях уровень управления отделен от устройств передачи данных: коммутаторы не участвуют в определении маршрутов для пакетов, а только реализуют программу контроллера. Основным стандартом, применяемым для построения ПКС, является протокол OpenFlow [2]. С помощью этого протокола реализуется независимый от производителя интерфейс между логическим контроллером и сетевыми коммутаторами ПКС.

Сеть OpenFlow состоит из коммутаторов, управляемых централизованным контроллером. Пакет, передаваемый по сети, обрабатывается контроллером существенно медленнее, нежели коммутатором, поэтому одной из основных функций контроллера является организация работы коммутаторов так, чтобы они обрабатывали большую часть пакетов, и лишь в исключительных случаях пакеты обрабатывались бы на контроллере.

Организация работы коммутаторов заключается в установке правил в таблицы коммутации (flow tables), определяющих, как будут обрабатываться те или иные пакеты. Правило состоит из шаблона, идентифицирующего вид пакетов, целочисленного приоритета, устраняющего неоднозначность

в случае наложения шаблонов, целочисленного лимита времени, указывающего число секунд до истечения срока активности правила, и списка действий, описывающих обработку пакета. Также для каждого правила в этих таблицах коммутатор содержит счетчики для учета количества и размера обрабатываемых пакетов.

Коммутатор обрабатывает приходящий пакет в три этапа. Сначала он выбирает правило из своей таблицы коммутации так, чтобы заголовок пакета соответствовал шаблону этого правила. Если такового не найдено, коммутатор отправляет пакет контроллеру для дальнейшей обработки. Иначе выбирается совпадающее по шаблону правило с наибольшим приоритетом. На втором шаге обновляются счетчики для выбранного правила. И, наконец, к пакету поочередно применяются все действия, записанные в правиле. К допустимым действиям относятся отправка пакета на порт и переписывание заголовочного поля правила.

Контроллер устанавливает правила в таблицы коммутации, реагируя на события в сети. Такими событиями являются подключение нового коммутатора к сети, удаление ранее действующего коммутатора из сети, события для сбора статистики, истечение лимита времени правила коммутатора. Также контроллер оперирует функциями отправки сообщений коммутаторам: установкой правил в таблицу коммутации, удалением всех правил с заданным шаблоном из таблицы коммутации, отправкой пакета и действия для его обработки коммутатором.

В статье исследуется возможность верификации ПКС как распределенных систем реального времени. Для этого потребовалось решить следующие задачи: выбрать адекватное средство для построения сетей; выбрать подходящее средство для верификации сетей как систем реального времени; построить корректный транслятор из выбранного средства построения сетей во входной язык нужной системы верификации; провести экспериментальное исследование возможности применения выбранного средства верификации для проверки спецификаций ПКС.

OpenFlow как стандарт реализации ПКС включает в себя большое количество свойств и предписаний для коммутации пакетов в сети. Производители компонентов компьютерных сетей используют лишь часть из них. Мы также ограничимся в рассматриваемых вариантах. При моделировании правил таблиц коммутации не будем уделять внимание приоритетам и счетчикам, а из всех допустимых действий будем рассматривать только действия типа отправки пакета на порт коммутатора. Сбор статистических данных в сети также не будет нас интересовать. С другой стороны, в нашу модель будут добавлены временные характеристики, описывающие работу физических объектов сети. Посредством этого мы будем учитывать не только предписания стандарта OpenFlow, но и технические возможности коммутаторов и каналов.

Статья обладает следующей логической структурой. Сначала мы опишем характеристики ПКС, учитываемые нами в предлагаемой модели, и общую схему верификации свойств ПКС. Здесь же дадим базовые сведения о выбранном инструменте верификации UPPAAL. Далее определим формальные синтаксис и семантику предложенной нами модели ПКС. После этого опишем разработанный нами алгоритм трансляции ПКС в сети временных автоматов, позволяющий проверять свойства ПКС в рамках возможностей UPPAAL. Затем представим теорему, обосновывающую корректность алгоритма трансляции, и приведем экспериментальные результаты, демонстрирующие возможности применения нашего подхода к верификации временных свойств ПКС.

Схема верификации

Общая схема верификации свойств ПКС состоит в следующем: сначала строится модель ПКС, учитывающая выбранные нами временные характеристики сети; затем эта модель корректным образом транслируется в сеть временных автоматов; далее свойство ПКС представляется в виде формулы логики LTL_{-X} с возможностью использования временных ограничений; построенная

формула автоматически проверяется на модели средством UPPAAL. Остаток главы посвящен пояснению основных особенностей модели и краткому описанию средства UPPAAL.

Модель имеет три составляющие: модель коммутатора, модель контроллера и описание сетевых каналов. В модели коммутатора учтены и описываются следующие временные характеристики: *rule_imp* – задержка поиска и выполнения правила на коммутаторе (временной интервал); *rule_def* – задержка установки в коммутатор правила от контроллера; *rule_ar* – интенсивность поступления пакетов из внешней среды; *rule_con* – задержка доставки необработанного пакета контроллеру. Также описываются физические характеристики *port* – множество портов коммутатора, соединяющих его с другими коммутаторами и внешней средой, и *tab* – объем таблицы коммутации, т. е. максимальное число правил таблицы.

Сетевые каналы описываются тем, какие порты они соединяют, и тем, какие временные ограничения на доставку пакетов они имеют. Поведение контроллера определяется политикой коммутации пакетов в сети, которая для нашего класса задач представима в виде таблицы, отображающей соответствие между заголовком поступившего на контроллер пакета и правилом, которое необходимо установить на коммутатор, приславший этот пакет. Модель контроллера состоит из множества записей такой таблицы.

В следующих разделах приведено формальное описание всех составляющих предложенной нами модели ПКС и их поведения, пригодное для применения формальных математических методов анализа.

Для построения модели ПКС был выбран кроссплатформенный редактор диаграмм Dia [3]. Модель сети из рассматриваемого нами подкласса описывается с помощью объектов библиотеки элементов UML, предоставляемых редактором, как показано на рис. 1.

Каждый коммутатор изображается в виде прямоугольника с тремя секциями. В верхней секции обозначается имя коммутатора, в нижней содержатся правила таблицы коммутации, в средней – описанные выше физические характеристики коммутатора. Каждое правило таблицы коммутации имеет четыре поля: имя порта, на который поступил пакет, заголовок поступившего пакета, лимит времени жизни правила и порт, на который необходимо отправить пакет.

Для отображения топологии сети ком-

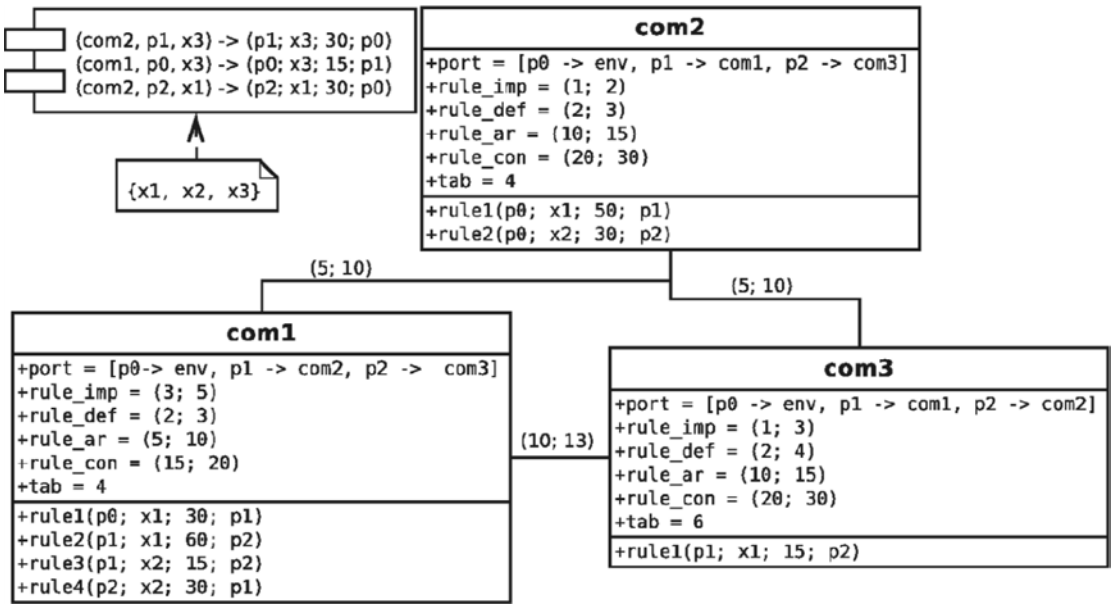

Рис. 1. Пример описания ПКС диаграммами Dia

мутаторы соединяются линиями, моделирующими дуплексные каналы сети. Контроллер изображается как прямоугольник с набором правил для коммутаторов. Также отдельно описывается множество всех возможных заголовков пакетов.

Для проверки спецификаций модели ПКС было выбрано программно-инструментальное средство UPPAAL [4]. Особенностью UPPAAL является возможность проверять свойства модели, в описание которой включено реальное время, это и определило наш выбор инструмента верификации. Для верификации UPAAL использует метод проверки свойств на моделях. Данный метод предполагает наличие модели, описывающей систему на определенном уровне абстракции, и позволяет проверить, удовлетворяет ли заданная модель системы формальным спецификациям. В средстве верификации UPPAAL в качестве модели используются сети конечных временных автоматов (параллельные композиции временных автоматов) [5], а формальные спецификации задаются формулами темпоральной логики TCTL [4].

Временные автоматы представляют собой конечные автоматы, работающие в реальном времени и осуществляющие синхронизацию посредством передачи сигналов через каналы связи. Особенностью таких автоматов является возможность использования таймеров. Значения таймеров можно указывать во временных ограничениях условий переходов между состояниями. Показания всех таймеров изменяются на одинаковые величины с течением времени.

Для верификации ПКС, описанных в терминах диаграмм Dia, средством UPPAAL нами предложен алгоритм трансляции диаграмм в сети временных автоматов. Сеть, получаемая при трансляции, состоит из автоматов, моделирующих коммутаторы, контроллер, внешнюю среду и каналы сети. Таким образом, в результате трансляции диаграмм мы получаем сеть, пригодную для верификации средством UPPAAL. Подобный подход к верификации распределенных систем реального времени применен в работе [6].

Формальный синтаксис ПКС

Перед описанием алгоритма трансляции необходимо предоставить формальное описание всех компонентов сети ПКС. С учетом выбранных нами ограничений ПКС может быть описана системой (H, con, Com, $Chan$), где H – множество заголовков пакетов сети, con – контроллер, $Com = (com_1, ..., com_n)$ – набор коммутаторов сети и $Chan = (c_1, ..., c_m)$ – набор каналов сети. Заметим, что во введенной формальной модели ПКС вместо пакетов рассматриваются только их заголовки, при этом содержащиеся в пакетах сообщения опускаются. Для простоты восприятия далее под пакетами в формальной модели будут подразумеваться их заголовки. Здесь и далее символы L и R будут использоваться для обозначения соответственно левой и правой границы интервалов, описывающих временные характеристики сети, такие как время доставки и обработки пакетов.

Коммутатор com_i включает в себя множество портов $Port_i$, начальную таблицу коммутации $Rule_i \subseteq Port_i \times H \times Real \times Port_i$ объема tab и временные характеристики L_i^{imp}, R_i^{imp}, L_i^{def}, R_i^{def}, L_i^{ar}, L_i^{con}, R_i^{con}, естественным образом соотносящиеся с характеристиками коммутатора, описанными в диаграмме (см., например, рис. 1). Правило (p, h, x, p') таблицы коммутации означает, что пакет h, пришедший на порт p, перенаправляется на порт p'. При этом x – время жизни правила; если $x \geq 0$, то правило назовем активным, иначе – истекшим. Шаблоном правила (p, h, x, p') будем называть пару (p, h).

Контроллер con представляет собой множество пар вида (i, r), означающих, что правило r может быть выслано коммутатору com_i. Канал c описывается тем, из какого порта какого коммутатора он исходит, и какими временными характеристиками задержки доставки пакетов (L, R) обладает. Заметим, что в предложенной модели каналы являются однонаправленными. Такое ограничение несущественно, т. к. дуплексный канал моделируется двумя однонаправленными.

Формальная семантика ПКС

Для доказательства корректности алго-

ритма трансляции необходимо ввести формальное описание работы ПКС. Данное описание является формализацией поведения ПКС в рамках стандарта OpenFlow с учетом введенных нами ограничений.

Состояние ПКС складывается из состояний контроллера и всех коммутаторов и каналов. Для удобства описания предполагаем, что каждый коммутатор com_i располагает следующими каналами: канал c_i^{ar} входного потока, по которому поступают пакеты из окружения, с временными характеристиками $L = L_i^{ar}$, $R = R_i^{ar}$, ведущий в специально выделенный под него порт; канал c_i^{tocon}, ведущий в контроллер из другого, специально выделенного порта, с характеристиками $L = L_i^{con}$, $R = R_i^{con}$; канал $c_i^{fromcon}$, ведущий от контроллера в третий, специально выделенный порт коммутатора, с характеристиками $L = R = 0$. Таким образом, состояние S ПКС включает в себя состояния s^{con} контроллера, $s_1^{com}, ..., s_n^{com}$ коммутаторов и $s_1^{ch}, ..., s_m^{ch}$, $s_1^{ar}, ..., s_n^{ar}$, $s_1^{tocon}, ..., s_n^{tocon}$, $s_1^{fromcon}, ..., s_n^{fromcon}$ каналов *ch* между коммутаторами, *ar* от входного потока, *tocon* к и *fromcon* от контроллера соответственно.

Коммутатор com_i может находиться в состояниях (*start*, *Rule*), (*select*, *t*, *Rule*, *h*, *p*), (*hit*, *Rule*, *h*, *p*), (*miss*, *Rule*, *h*, *p*), (*mod*, *t*, *Rule*). В состоянии *start* коммутатор прослушивает входящие каналы на предмет поступивших пакетов. В состоянии *select* коммутатор, получив на порт *p* пакет *h*, просматривает таблицу коммутации для принятия решения о перенаправлении пакета. В состоянии *hit* пакет засылается в канал, исходящий из порта *p*. В состоянии *miss* шаблон, состоящий из пакета *h* и порта *p*, на который он пришел, засылается в канал к контроллеру. Вообще говоря, коммутатор также должен посылать контроллеру свой идентификатор, однако в построенной формальной модели идентификатор может быть восстановлен по номеру порта, входящего в коммутатор. В состоянии *mod* происходит запись в таблицу коммутации правила, присланного контроллером. Во всех состояниях коммутатора *Rule* – текущая таблица коммутации, *t* – время нахождения в текущем управляющем состоянии, *h* и *p* – хранимые пакет и порт.

Каналы могут находиться в состояниях (*empty*), (*full*, *t*, *o*) и (*sent*, *o*), где *o* – пакет (для обычных и поточных каналов), шаблон (для каналов к контроллеру) либо правило (для каналов от контроллера). Компонента $t \in Real$ – время обработки сообщения каналом. Состояние *empty* соответствует пустому каналу. В состоянии *full* в канале содержится пока ещё не доставленный пакет. В состоянии *sent* канал содержит доставленный пакет и ожидает считывания этого пакета коммутатором или контроллером.

Контроллер может находиться только в состояниях (*idle*) (ожидание запросов от коммутаторов) и (*send*, *i*, *r*) (послать *i*-му коммутатору новое правило *r*).

Начальное состояние S_o системы строится из начальных состояний коммутаторов, контроллера и каналов. Начальное состояние *i*-го коммутатора – (*start*, $Rule_i$), контроллера – (*idle*), каналов – (*empty*).

Работа ПКС характеризуется последовательностью состояний, начинающейся в S_0 и строящейся по описанным далее правилам. Запись $s \to s'$ означает, что следующее состояние может быть получено из предыдущего заменой компоненты *s* на *s'*. Запись $s_1, s_2 \to s_1', s_2'$ означает то же самое для пары компонент. Построение вычисления ПКС происходит по следующим правилам.

1. Получение пакета: s_i^{com}, s_i^{ar} = (*start*, *Rule*), (*sent*, *h*) → (*select*, 0, *Rule*, *h*, *p*), (*empty*).

2. Применение активного правила $r = (p, h, x, p')$: s_i^{com} = (*select*, *Rule*, *t*, *h*, *p*) → (*hit*, *Rule*, *h*, *p'*), если $t \in [L_i^{imp}, R_i^{imp}]$, $r \in Rule$.

3. Обработка истекшего правила $r = (p, h, x, p')$: s_i^{com} = (*select*, *Rule*, *t*, *h*, *p*) → (*miss*, *Rule'*, *h*, *p*), если $t \in [L_i^{imp}, R_i^{imp}]$, $r \in Rule$ и $Rule' = Rule \setminus \{r\}$.

4. Сброс пакета: s_i^{com} = (*select*, *Rule*, *t*, *h*, *p*) → (*start*, *Rule*), если $t \in [L_i^{imp}, R_i^{imp}]$, $|Rule| = tab_i$, все правила из *Rule* активны и ни одно из них не имеет шаблона (*p*, *h*).

5. Несоответствие шаблонов: s_i^{com} = = (*select*, *Rule*, *t*, *h*, *p*) → (*miss*, *Rule'*, *h*, *p*), где *Rule'* получается из *Rule* удалением всех истекших правил.

6. Начало перезаписи таблицы: s_i^{com},

$s_i^{fromcon}$ = (*start*, *Rule*), (*sent*, *rule*) → (*mod*, 0, *Rule*), (*sent*, *rule*).

7. Успешная перезапись таблицы: s_i^{com}, $s_i^{fromcon}$ = (*mod*, *t*, *Rule*), (*sent*, (*p*, *h*, *x*, *p'*)) → (*hit*, *Rule'*, *h*, *p'*), (*empty*), если $t \in [L_i^{def}, R_i^{def}]$, |*Rule*| < tab_i и *Rule'* = *Rule* ∪ {(*p*, *h*, *x*, *p'*)}.

8. Неуспешная перезапись таблицы: s_i^{com}, $s_i^{fromcon}$ = (*mod*, *t*, *Rule*), (*sent*, *rule*) → (*start*, *Rule*), (*empty*), если $t \in [L_i^{def}, R_i^{def}]$ и |*Rule*| = *tab*.

9. Пересылка пакета коммутатору: s_i^{com}, s_j^{ch} = (*hit*, *Rule*, *h*, *p*), (*empty*) → (*start*, *Rule*), (*full*, 0, *h*), если канал c_j исходит из порта *p* коммутатора *i*.

10. Пересылка шаблона контроллеру: s_i^{com}, s_i^{tocon} = (*miss*, *Rule*, *h*, *p*), (*empty*) → (*start*, *Rule*), (*full*, 0, (*p*, *h*)).

11. Успешная обработка шаблона контроллером: s^{con}, s_i^{tocon} = (*idle*), (*sent*, (*p*, *h*)) → (*send*, *i*, *r*), (*empty*), если (*i*, (*p*, *h*, *x*, *p'*)) ∈ *con*.

12. Неуспешная обработка шаблона контроллером: s^{con}, s_i^{tocon} = (*idle*), (*sent*, (*p*, *h*)) → (*idle*), (*empty*), если (*i*, (*p*, *h*, *x*, *p'*)) ∉ *con*.

13. Пересылка правила от контроллера: s^{con}, $s_i^{fromcon}$ = (*send*, *i*, *r*), (*empty*) → (*idle*), (*full*, 0, *r*).

14. Появление пакета в сети: s_i^{ar} = = (*empty*) → (*full*, 0, *h*), если *h* ∈ *H*.

15. Сброс пакета в окружение: s_i^{com} = = (*hit*, *Rule*, *h*, *p*) → (*start*, *Rule*), если ни один канал c_j не исходит из порта *p* коммутатора com_i.

16. Доставка в канале: (*full*, *t*, *o*) → (*sent*, *o*), если *t* ∈ [*L*, *R*] для характеристик *L*, *R* соответствующего канала.

17. Продвижение времени: таймеры всех коммутаторов и каналов увеличиваются на одну и ту же положительную величину *d*, времена жизни правил уменьшаются на эту же величину, и при этом:

если какой-либо канал находится в состоянии (*full*, *t*, *o*), то $t + d \le R$ для характеристики *R* этого канала;

если s_i^{com} = (*select*, *Rule*, *t*, h, *p*), то $t + d \le R_i^{imp}$;

если s_i^{com} = (*mod*, *Rule*, *t*), то $t + d \le R_i^{def}$.

Заметим, что в каждый момент времени к текущему состоянию сети может быть применено в общем случае более одного правила. В такой ситуации недетерминированно может быть выбрано любое из правил. Роль окружения в нашей модели заключается в генерации пакетов для заданных портов коммутаторов и приеме предназначенных для сброса во внешнюю среду пакетов. Таким образом мы абстрагируемся от конкретного окружения.

Алгоритм трансляции

Алгоритм трансляции переводит ПКС в эквивалентную ей сеть временных автоматов. Уточнение понятия эквивалентности обсуждается в следующем разделе.

Сеть *N*, получаемая в результате трансляции ПКС ((*com*$_1$, ..., *com*$_n$), *con*, (*c*$_1$, ..., *c*$_m$), *H*), содержит автоматы *Hurry*, *Env*, *Stream*, *Chan*, A_{con} и A_i, $i \in \{1, ..., n\}$.

Каждый канал c_i моделируется булевыми переменными *full*[c_i] (пакет послан), *ready*[c_i] (пакет доставлен), таймером *t*[c_i] и переменными для хранения объектов, пересылаемых через канал. Сброс пакетов в окружение моделируется с помощью особого канала c_{env}. Также в сеть добавляются все каналы c_i^{ar}, c_i^{tocon}, $c_i^{fromcon}$.

Автомат *Hurry* обеспечивает срочные дуги (т. е. дуги, выполняющиеся немедленно по выполнении их предусловий) и состоит из одной вершины и петли, принимающей сигнал по срочному каналу *hurry*. К срочным дугам по умолчанию добавляется посылка сигнала по каналу *hurry*. Автомат *Env* удаляет пакеты из канала c_{env} и состоит из одной вершины и срочной петли с записью в переменную *full*[c_{env}] значения *false*. Автомат *Stream* генерирует пакеты, состоит из одной вершины и содержит набор срочных петель, недетерминированно засылающий пакеты в каналы c_i^{ar}. Автомат *Chan* обеспечивает доставку пакетов каналами, состоит из одной вершины и содержит петлю для каждого канала *c*, помеченную предусловием *full*[*c*]&&!*ready*[*c*]&&(*t*[*c*] ≥ *L*(*c*)) и записью в переменную *ready*[*c*] значения *true*, где *L*(*c*) – характеристика *L* канала *c*. Сама вершина автомата *Chan* при этом помечена инвариантом – конъюнкцией неравенств *t*[*c*] ≤ *R*(*c*).

Автомат A_{con} имеет два обычных состояния – *idle* и *send* – и одно срочное состоя-

ние *l*. Срочная дуга *idle* → *l* записывает в локальные переменные автомата шаблон, присланный коммутатором, и номер этого коммутатора и в зависимости от наличия отправляемого обратно правила переходит либо обратно в *idle*, либо в *send*. Срочная дуга *send* → *idle* присваивает переменной *full*[*c*] значение *true* и записывает в переменную канала выбранное правило.

Автомат A_i моделирует работу коммутатора com_i и состоит из обычных состояний *start*, *select*, *hit*, *miss*, *mod*, соединенных через срочные вершины. Срочная дуга *start* → *select* записывает в локальные переменные автомата доставленный через канал *c* пакет и порт, на который он пришел, и записывает в переменную *full*[*c*] значение *false*. Состояние *select*, помеченное инвариантом $t \le R_i^{imp}$, имеет одну исходящую дугу в срочное состояние *l*, помеченную предусловием $t \ge L_i^{imp}$. Здесь *t* – локальный таймер коммутатора. Из состояния *l* в зависимости от наличия шаблона происходит переход в состояния *hit* и *miss* с модификацией таблицы согласно правилам 2–5 семантики ПКС. Удаление многих правил из таблицы коммутации обеспечивается срочной вершиной *l'* с петлей, удаляющей из таблицы истекшие правила, и исходящей дугой, помеченной предусловием, утверждающим, что из таблицы удалены все истекшие правила. Срочная дуга *start* → *mod* записывает в локальные переменные автомата правило, доставленное каналом $c_i^{fromcon}$. Состояние *mod* помечено инвариантом $t \le R_i^{def}$, исходящие из него дуги – предусловием $r \ge L_i^{def}$. В зависимости от того, заполнена ли таблица коммутации, из состояния *mod* автомат может либо перейти в состояние *start*, либо перейти в состояние *hit* с записью правила в таблицу.

Описанный в этом разделе алгоритм трансляции будем обозначать записью *Alg*, а сеть временных автоматов, получаемую из ПКС-модели *N*, – записью *Alg*(*N*).

Корректность алгоритма трансляции

Важным результатом данной статьи является строгое обоснование корректности алгоритма *Alg*, – эквивалентности по прореживанию (stuttering equivalence) систем переходов [7], описывающих поведение исходной ПКС *N* и сети временных автоматов *Alg*(*N*), получаемой в результате применения алгоритма *Alg*. Обоснования такой эквивалентности достаточно для того, чтобы признать все формулы LTL_{-X}, а значит, и все формулы, которые могут быть проверены средством UPPAAL [4], равновыполнимыми.

Под корректностью алгоритма *Alg* понимается равновыполнимость формул, используемых в средстве UPPAAL, для исходной ПКС *N* и результирующей сети *Alg*(*N*). Ключевым понятием в доказательстве корректности алгоритма *Alg* является эквивалентность по прореживанию (stuttering equivalence) для систем переходов [7]. Неформально такая эквивалентность означает, что если в каждом вычислении двух данных систем склеить все одинаковые состояния, то мы получим одинаковые множества вычислений. Одинаковыми полагаются состояния с совпадающими множествами выполнимых формул. В работе [8] сформулирована следующая теорема.

Теорема 1. Если системы переходов M_1, M_2 эквивалентны по прореживанию и формула Ф логики LTL_{-X} истинна для M_1, то она также истинна для M_2.

Следующая теорема позволяет применить только что сформулированную к системам переходов, описывающим поведение ПКС *N* и сети *Alg*(*N*). Система переходов, описывающая поведение сети временных автоматов, обсуждается, например, в [5]. Пусть TS_A – система переходов, описывающая поведение системы *A*.

Теорема 2. Пусть *N* – произвольная ПКС. Тогда системы переходов TS_N и $TS_{Alg(N)}$ эквивалентны по прореживанию.

Теорему обосновывают следующие рассуждения. Состояниям контроллера и коммутаторов ставятся в соответствие состояния автоматов, в которые они транслируются, составляемые из одноименных управляющих состояний и из значений локальных переменных, представленных в описании алгоритма трансляции. Состояниям каналов ставятся в соответствие наборы значений переменных *full*, *ready* и переменных для хранения данных. Одному применению

семантических правил 1–17 соответствует последовательность переходов в сети временных автоматов через срочные состояния, и смена значений переменных происходит только в одном месте последовательности. Если к состоянию ПКС N применять правило из описания формальной семантики и к соответствующему состоянию сети $Alg(N)$ применять описанную выше последовательность переходов, то результирующее состояние ПКС N будет соответствовать результирующему состоянию сети $Alg(N)$.

Корректность алгоритма напрямую следует из приведенных теорем с учетом того, что формулы, проверяемые средством UPPAAL, могут быть переформулированы в терминах логики LTL_{-X}.

Экспериментальное исследование

На рис. 2–4 приведена сеть временных автоматов, полученная в результате трансляции из ПКС, изображенной на рис. 1. В целях удобочитаемости выражения автоматов написаны в синтаксисе, отличном от синтаксиса UPPAAL и при этом более простом для восприятия.

Запись вида (i = 1..3, 5, 7) означает перебор всех i из заданного диапазона и создание копий дуги для каждого значения i. Запись вида (i = 1..3 → Ф) означает «для всех i из заданного диапазона верна формула Ф».

Функция *get*(c) сохраняет содержимое канала c в локальные переменные (h, p, ...) и выполняет присваивание c = *false*. Функция *send*(c, o) копирует o в содержимое канала и выполняет присваивания c = *true*, *c_ready* = *false*, *c_t* = 0. Функция *set_rule*(i) копирует локальные переменные коммутатора, отвечающие компонентам правила, в i-ю позицию таблицы.

Предикат *to_deliver*(c) описывается как $c\&\&!c\_ready\&\&(c\_t \geq c\_L)$. Предикат *ok*($c$) описывается как $!c||c\_ready||(c\_t \leq c\_R)$.

Канал c[0] выделен под окружение, каналы c[1], c[2], c[3] – под потоки, прикрепленные к соответствующим коммутаторам. Автоматы A_2, A_3 отличаются от автомата A_1 только номерами задействованных каналов и константами, определяемыми количеством портов и объемом таблицы, и по этой причине опущены.

Заметим, что в наши задачи входила не верификация «реальных» сетей, содержащих сотни коммутаторов и тысячи правил в таблицах коммутации (очевидно, это невозможно на начальной стадии исследования), а описание главных идей верификации временных аспектов поведения ПКС. Ниже приведены проверенные с помощью средства UPPAAL свойства и их запись на языке запросов UPPAAL [4].

1. В работе системы не возникает блокировки:

$$A[]\ not\ deadlock$$

2. В сеть всегда будут поступать пакеты из внешней среды:

$$A<>\ forall(num : int\ [0,2])\ (channel\_h[stream.align[num]])$$

3. Допустим сценарий работы сети, в котором коммутатор не принимает ни одного пакета:

$$E[]\ com1.start$$

4. При любом сценарии работы сети контроллер обработает хотя бы один пакет:

$$E<>\ !con.idle$$

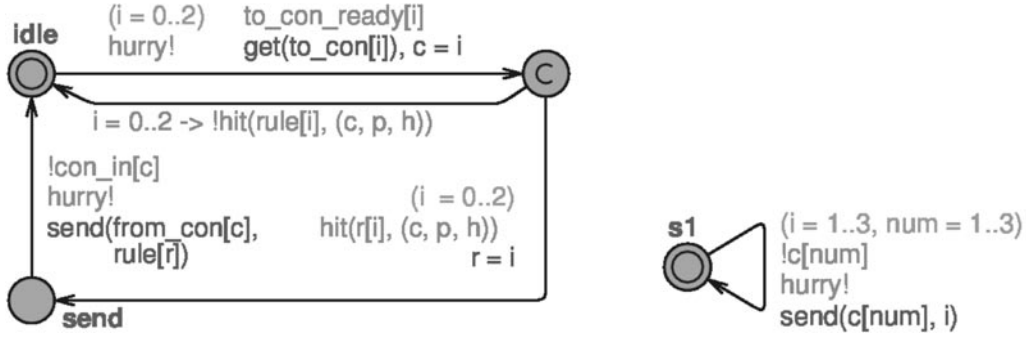

Рис. 2. Автоматы A_{con} (слева), *Stream* (справа)

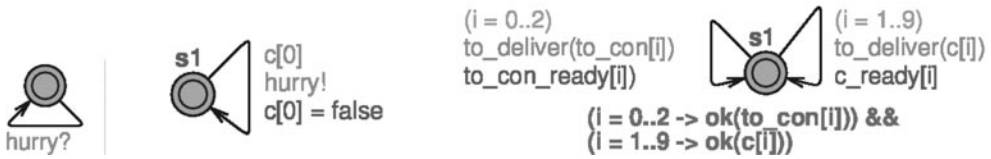

Рис. 3. Автоматы *Hurry* (слева), *Env* (в центре), *Chan* (справа)

5. Хотя бы один пакет будет успешно перенаправлен коммутатором (т. е. коммутатор выполняет свою главную функцию):

$$E<>\ com1.hit$$

Свойства 1, 2, 4, 5 соответствуют спецификации сетей OpenFlow, в то время как свойство 3 описывает нежелательные сценарии работы сети. Проверка показала, что свойства 1, 2, 4, 5 выполняются, а свойство 3 не выполняется. Это свидетельствует о том, что функционирование полученной сети временных автоматов соответствует нашим ожиданиям и что предложенная схема верификации ПКС с помощью средства UPPAAL позволяет проверять свойства поведения ПКС как системы реального времени.

В ячейках таблицы представлено время проверки описанных темпоральных свойств для некоторых моделей ПКС: 1 – три коммутатора в топологии кольца (см. рис. 1); 2 – четыре коммутатора в топологии звезды; 3 – четыре коммутатора в произвольной топологии; 4 – два коммутатора с изначально пустыми таблицами коммутации. Проверка проводилась на вычислительном устройстве со следующими характеристиками: INTEL Core i7 3820/1600 МГц/2Гб DDR3. Первое свойство не было проверено на моделях 1–3 в связи с нехваткой памяти.

В результате проведенных исследований мы подтвердили практическую осуществимость подхода, предложенного в статье [6], для проверки выполнимости свойств поведения моделей программно-конфигурируемых сетей в реальном времени. Предложенное нами средство анализа поведения ПКС может быть использовано для наглядного описания конфигурации и коммутационной политики сети в виде диаграмм. Разработанный нами алгоритм трансляции преобразует диаграмму в сеть временных автоматов, подаваемую на вход системы верификации

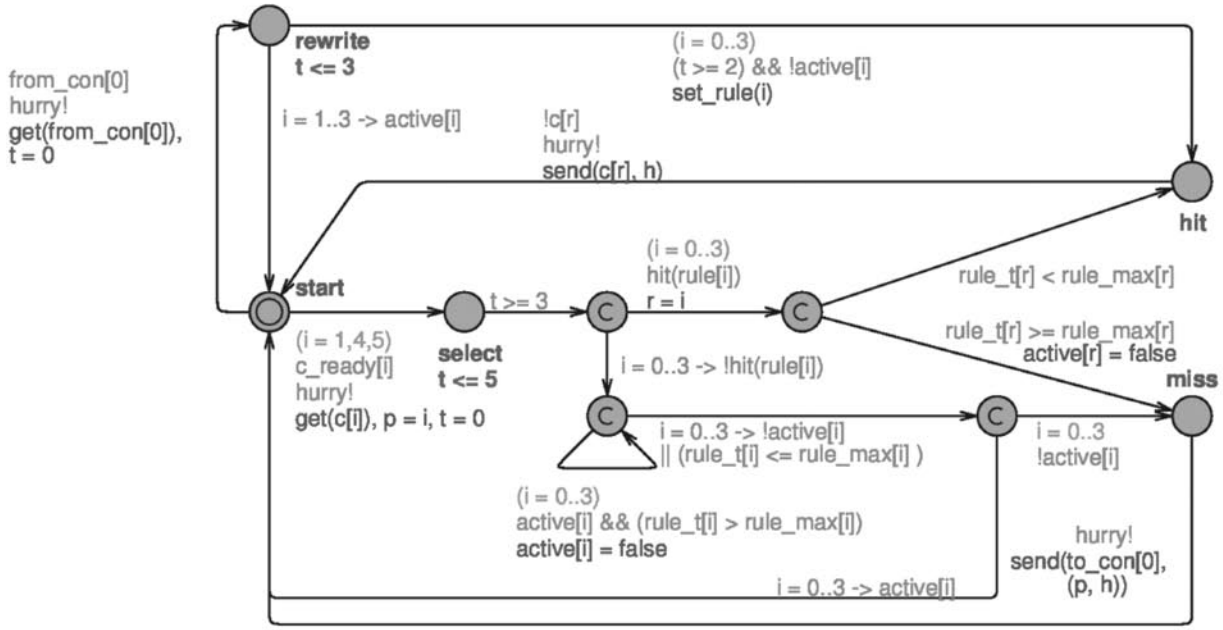

Рис. 4. Автомат A_1

Время проверки свойств ПКС

	Свойство				
	1	2	3	4	5
Модель 1	–	1 с	1 с	7 с	1 с
Модель 2	–	1 с	1 с	1 мин 2 с	1 мин 25 с
Модель 3	–	1 с	1 с	1 мин	1 мин 19 с
Модель 4	27 ч	1 с	1 с	1 с	1 с

UPPAAL. Корректность алгоритма трансляции строго обоснована, транслятор реализован на языке программирования Perl. Получаемая на выходе транслятора сеть временных автоматов позволяет проверять спецификации ПКС с помощью системы UPPAAL. Особенностью такой верификации является возможность учитывать временной аспект поведения ПКС как распределенных систем реального времени.

СПИСОК ЛИТЕРАТУРЫ

1. **Casado M., Garfinkel T., Akella A., Fredman M., Boneh D., McKeown N., Shenker S.** SANE: A Protection Architecture for Enterprise Networks // 15th Usenix Security Symp. Vancouver, Canada, 2006.

2. **McKeown N., Anderson T., Balakrishnan H., Parulkar G., Peterson L., Rexford J., Shenker S., Turner J.** Openflow: Enabling innovation in campus network // SIGCOMM Computer Communication Review. 2008. Vol. 38. No. 2. Pp. 69–74.

3. Dia [электронный ресурс]/ URL: http://dia-installer.de/doc/en/dia-manual.pdf (дата обращения 08.11.2013)

4. **Behrmann G., David A., Larsen K.** A tutorial on Uppaal // Lecture Notes in Computer Science. 2004. Vol. 3185. Pp. 200–236.

5. **Alur R., Dill D.** Automata for modelling real-time systems // Proc. of Internat. Colloquium on Algorithms, Languages, and Programming, LNCS. 1990. Vol. 443. Pp. 322–335.

6. **Волканов Д.Ю., Захаров В.А., Зорин Д.А., Коннов И.В., Подымов В.В.** Как разработать простое средство верификации систем реального времени // Моделирование и анализ информационных систем. 2012. № 6. Т. 19. С. 45–56.

7. **Browne M.C., Clarke E.M., Grumberg O.** Characterizing finite Kripke structures in propositional temporal logics // Theor. Comp. Sci. 1988. Vol. 59(1-2). Pp. 115–131.

8. **Clarke E.M., Grumberg O., Peled D.** Model Checking. The MIT Press, 1999.

REFERENCES

1. **Casado M., Garfinkel T., Akella A., Fredman M., Boneh D., McKeown N., Shenker S.** SANE: A Protection Architecture for Enterprise Networks. *15th Usenix Security Symposium*, Vancouver, Canada, August 2006.

2. **McKeown N., Anderson T., Balakrishnan H., Parulkar G., Peterson L., Rexford J., Shenker S., Turner J.** Openflow: Enabling innovation in campus network. *SIGCOMM Computer Communication Review*, 2008, Vol. 38, No. 2. Pp. 69–74.

3. Dia. Available: http://dia-installer.de/doc/en/dia-manual.pdf (Accesed 08.11.2013)

4. **Behrmann G., David A., Larsen K.** A tutorial on Uppaal, *Lecture Notes in Computer Science*, 2004, Vol. 3185. Pp. 200–236.

5. **Alur R., Dill D.** Automata for modelling real-time systems, *Proc. of Internat. Colloquium on Algorithms, Languages, and Programming,* LNCS, 1990, Vol. 443, Pp. 322–335.

6. **Volkanov D.Yu., Zakharov V.A., Zorin D.A., Konnov I.V., Podymov V.V.** Kak razrabotat prostoye sredstvo verifikatsii sistem realnogo vremeni [On the Designing of Model Checkers for Real-Time Distributed Systems], *Modelirovaniye i analiz informatsionnykh system* [*Modelling and Analysis of Information Systems*], 2012, No. 6, Vol. 19, Pp. 45–56.

7. **Browne M.C., Clarke E.M., Grumberg O.** Characterizing finite Kripke structures in propositional temporal logics, *Theor. Comp. Sci.*, 1988, Vol. 59(1-2), Pp. 115–131.

8. **Clarke E.M., Grumberg O., Peled D.** *Model Checking*. The MIT Press, 1999.

ПОДЫМОВ Владислав Васильевич – *аспирант факультета вычислительной математики и кибернетики Московского государственного университета имени М.В. Ломоносова.*
119991, Россия, Москва, ГСП-1, Ленинские горы, д. 1, стр. 52.
E-mail: valdus@yandex.ru

PODYMOV, Vladislav V. *Lomonosov Moscow State University.*
119991, GSP-1, Leninskie Gory, Moscow, Russia.
E-mail: valdus@yandex.ru

ПОПЕСКО Ульяна Владиславовна – *аспирант факультета вычислительной математики и кибернетики Московского государственного университета имени М.В. Ломоносова.*
119991, Россия, Москва, ГСП-1, Ленинские горы, д. 1, стр. 52.
E-mail: ulya_kiber@mail.ru

POPESKO, Uliana V. *Lomonosov Moscow State University.*
119991, GSP-1, Leninskie Gory, Moscow, Russia.
E-mail: ulya_kiber@mail.ru